

REMARKS

I. Status of case

Claims 7-29 are currently pending in this case. Claims 7, 16 and 23 are independent claims.

II. Rejections under 35 U.S.C. §§ 102 and 103

Claims 7, 8 and 10-29 were rejected under 35 U.S.C. §102(b) as being anticipated by Colburn et al., (U.S. Patent No. 6,173,404). Claim 9 was rejected under 35 U.S.C. §103(a) as being unpatentable over Colburn.

Applicants respectfully contend that the Colburn reference does not teach or suggest the claims as currently presented. The Colburn reference includes the following Figures 7a-b and 8:

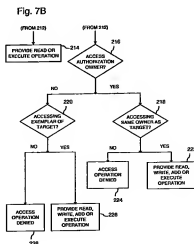
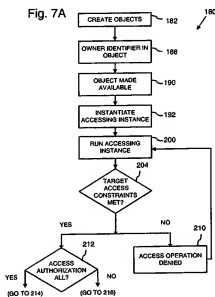
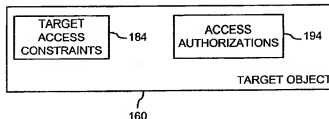


Fig. 8

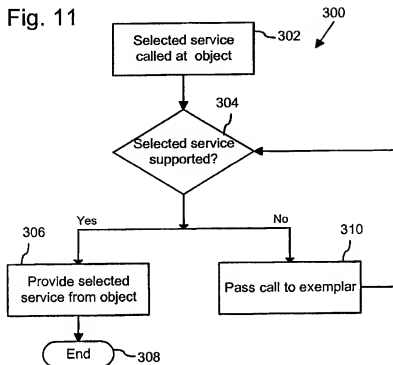


As evident from Figures 7a-b (and the associated text) and figure 8, Colburn teaches that the target object includes access constraints 184 and access authorizations 194. Further, Colburn teaches that the object is created before any access of any program. In particular, the objects and the associated access constraints are created prior to any instance of an access.

For example, the Colburn reference teaches that an object can have three different types of access constraints: All, Owner, and Exemplar. “All” means that anyone may access the object. When a user wishes to access the target object, the Colburn reference teaches that the access constraints (which were previously created for the object) are merely checked to determine whether to allow access. In the example given, if the object has an “All” access constraint, the user is given access.

The Colburn reference teaches that the objects may be dynamically updated at run-time (“dynamic inheritance”). Specifically, the Colburn reference teaches dynamic inheritance in Figure 11 (reproduced below):

Fig. 11

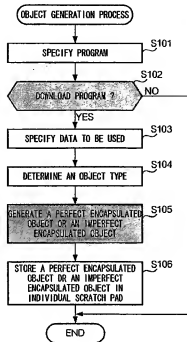


As taught in Colburn, the dynamic inheritance allows the object to change based on the service selected. Thus, while the Colburn reference discusses updating an object, Colburn fails to teach a process of generating such an object as disclosed in Figure 8 (such as a process of setting access constraints or access authorizations as disclosed in Figure 8).

In contrast to the Colburn reference, claim 7 recites:

an object generation manager that determines whether to generate a perfect encapsulated object or an imperfect encapsulated object based on reliability of one application program among the one or more application programs

Thus, the object generation manager in claim 7 generates a perfect encapsulated object or an imperfect encapsulated object based on the application program being executed – namely the “reliability” of the application program being executed. An example of this is disclosed in Figure 8 of the present application (reproduced below with highlights):



In this way, the generation of the perfect or imperfect encapsulated object is dependent on the program accessed. For example, when a program is accessed, it is determined whether it is downloaded from the Internet. If so, it is determined to be potentially unreliable, thereby requiring the generating of a perfect encapsulated object in order to protect the data. In this way, the generating of the perfect encapsulated object may only need to be performed when it is necessary.

This is in contrast to the Colburn reference in several respects. First, as discussed above, Colburn is silent as to how the object is generated. While Colburn discusses “dynamic inheritance”, Colburn is silent regarding how the object is generated. Second, Colburn does not even suggest that the object is generated based on the application accessed (let alone based on the reliability of the application accessed). Instead, the Colburn reference merely takes security measures regardless of whether it is needed or not. The Office Action contends that that “dynamic inheritance” relates to generating the object. Applicants respectfully disagree. The updating using “dynamic inheritance” is not based on the application accessed – including the “reliability” of the application accessed. And, the “dynamic inheritance” does not relate to “generating” the object. At best, the “dynamic inheritance” simply discloses what the updating of the object will be (i.e., how the object will be changed); “dynamic inheritance” does not teach anything about when the “generating” will occur (i.e., it does not teach that when the reliability

of the executed application is in question, the object will be updated). For these reasons alone, claim 7 as currently presented distinguish over the cited references.

Claim 16 recites the following:

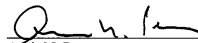
determine whether to generate an imperfect encapsulated object or a perfect encapsulated object based on the analysis of the application program or the data set

See also claim 23. Thus, the generated of an imperfect or perfect encapsulated object is based on analyzing the application program or the data set. As discussed above, the Colburn reference fails to teach any way of generating the object. And, the Colburn reference fails to teach any generating of the object based on the application (see above) or based on any data set. Therefore, claims 16 and 23 are patentable over the cited references.

SUMMARY

If any questions arise or issues remain, the Examiner is invited to contact the undersigned at the number listed below in order to expedite disposition of this application.

Respectfully submitted,



Amir N. Penn
Registration No. 40,767
Attorney for Applicant

BRINKS HOFER GILSON & LIONE
P.O. BOX 10395
CHICAGO, ILLINOIS 60610
(312) 321-4200